```
"""
Date: 2020. 06. 16.
Programmer: MH
Description: Code for Corona Map based on Geopandas library and GADM data
"""
import geopandas as gpd
import matplotlib.pyplot as plt
import pandas as pd
from datetime import datetime, timedelta

from shapely.geometry import Point, shape


class CoronaMap:

    def __init__(self):
        self.shapefile = {"South Korea": r".\gadm36_KOR_shp\gadm36_KOR_1.shp",
                          "South Korea 2": r".\gadm36_KOR_shp\gadm36_KOR_2.shp",
                          "Seoul": r"D:\2.
                          Project\Python\CoronaMap\201912기초구역DB_전체분\서울특별시\TL_KODIS_
                          BAS_11.shp"}
        self.corona_csv_path = r".\kr_regional_daily.csv"
        self.geometries = {}
        self.scale = 1.1    # Zoom scale
        self.ko_corona_data = None

    def displayCoMap(self, region, regionUnit, areaColor, numberFlag,percentageFlag,
    convertionFlag):
        """
        To display corona map
        :param region: string, the target place
        :param regionUnit: string, the unit of the sub-region; city, district
        :param areaColor: string, color map
        :param numberFlag: boolean, whether a number associated with each infected
        area should be displayed.
        :param percentageFlag: boolean, whether a percentage (%) associated with
        each infected area should be displayed.
        :param convertionFlag: boolean, whether a notational/color convention should
        be displayed on a corner of the map.
        :return:
        """
        self.region_geo_data = gpd.read_file(self.shapefile[region])   # To load geo
        data following "region"
        if region == "South Korea" and regionUnit=="District":
            self.region_geo_data = gpd.read_file(self.shapefile["South Korea 2"])
            # To load geo data following "region"
        self._preprocess_geodata(region)
        if region == "South Korea":
            self.load_corona_data(regionUnit)
        if self.ko_corona_data is not None:
            self.merge_data()

        self.fig, self.ax = plt.subplots(1, 1, figsize=(16, 8))
        print(self.region_geo_data.head())
        for i, row in self.region_geo_data.iterrows():
            polygon = row['geometry']
            centroid = polygon.centroid
            self.ax.text(centroid.x, centroid.y, row['region'], fontsize=13)    # To
            print region Name
            if numberFlag:
                self.ax.text(centroid.x+0.05, centroid.y-0.05, row['confirmed'],
                fontsize=10)
            if percentageFlag:
                self.ax.text(centroid.x+0.25, centroid.y-0.05, row['released'],
                fontsize=10)

        zp = ZoomPan()      # To add mouse wheel zoom
        zp.zoom_factory(self.ax, base_scale=self.scale)
        zp.pan_factory(self.ax)
        self.region_geo_data.plot(ax=self.ax, legend=convertionFlag, cmap=areaColor)

```

```
61          cid = self.fig.canvas.mpl_connect("button_press_event", self.on_click_region)
               # on click event
62          plt.show() # To show image
63
64      def on_click_region(self, event):
65          """
66          To define click event
67          :param event:
68          :return:
69          """
70          point = Point(event.xdata, event.ydata)
71          for region in self.geometries:
72              if point.within(shape(self.geometries[region])):
73                  print(region)
74                  self.fig, self.ax = plt.subplots(1, 1, figsize=(16, 8))
75                  self.region_geo_data = gpd.read_file(self.shapefile[region])   # To
                    load geo data following "region"
76                  zp = ZoomPan()      # To add mouse wheel zoom
77                  zp.zoom_factory(self.ax, base_scale=self.scale)
78                  zp.pan_factory(self.ax)
79                  self.region_geo_data.plot(ax=self.ax, legend=True, cmap="Reds")
80
81                  cid = self.fig.canvas.mpl_connect("button_press_event", self.
                    on_click_region)   # on click event
82                  plt.show() # To show image
83
84      def _preprocess_geodata(self, region):
85          """
86          To choose target columns from loaded geo data
87          :param region: string, input region
88          :return:
89          """
90          if region == "South Korea":  # if the target is South Korea
91              self.region_geo_data = self.region_geo_data[["NAME_1", "geometry"]]
92              self.region_geo_data.columns = ["region", "geometry"]
93              for i, row in self.region_geo_data.iterrows():
94                  self.geometries[row['region']] = row['geometry']
95          elif region == "Seoul":
96              self.region_geo_data.geometry = self.region_geo_data.buffer(0.001)
97              self.region_geo_data = self.region_geo_data.dissolve(by="SIG_CD")
98              self.region_geo_data = self.region_geo_data[["SIG_KOR_NM", "geometry"]]
99              self.region_geo_data.columns = ["region", "geometry"]
100
101
102     def load_corona_data(self, regionUnit):
103         """
104         To load corona data from csv file
105         :return:
106         """
107         if regionUnit == "City":
108             ko_whole_corona_data = pd.read_csv(self.corona_csv_path)
109             target_date = datetime.now().strftime("%Y%m%d")     # To set target date
                to today
110             self.ko_corona_data = ko_whole_corona_data[ko_whole_corona_data['date']
                == int(target_date)]    # To get today's data
111             if self.ko_corona_data.empty: # if there is no today's data
112                 target_date = (datetime.now()-timedelta(days=1)).strftime("%Y%m%d")
                    # To set  target date to yesterday
113                 self.ko_corona_data = ko_whole_corona_data[ko_whole_corona_data[
                    'date'] == int(target_date)]  # To load data
114         elif regionUnit == "District":
115             pass
116
117     def merge_data(self):
118         """
119         To merge geo data and corona data
120         :return: None
121         """
122         self.ko_corona_data = self.ko_corona_data.loc[:, ["region", "confirmed",
            "death", "released"]]
123         self.region_geo_data = self.region_geo_data.merge(self.ko_corona_data, on=
```

```
            "region", how="outer").dropna()


    def updateCoMap(self):
        pass

class ZoomPan:
    def __init__(self):
        self.press = None
        self.cur_xlim = None
        self.cur_ylim = None
        self.x0 = None
        self.y0 = None
        self.x1 = None
        self.y1 = None
        self.xpress = None
        self.ypress = None

    def zoom_factory(self, ax, base_scale = 2.):
        def zoom(event):
            cur_xlim = ax.get_xlim()
            cur_ylim = ax.get_ylim()

            xdata = event.xdata # get event x location
            ydata = event.ydata # get event y location

            if event.button == 'up':
                # deal with zoom in
                scale_factor = 1 / base_scale
            elif event.button == 'down':
                # deal with zoom out
                scale_factor = base_scale
            else:
                # deal with something that should never happen
                scale_factor = 1

            new_width = (cur_xlim[1] - cur_xlim[0]) * scale_factor
            new_height = (cur_ylim[1] - cur_ylim[0]) * scale_factor

            relx = (cur_xlim[1] - xdata)/(cur_xlim[1] - cur_xlim[0])
            rely = (cur_ylim[1] - ydata)/(cur_ylim[1] - cur_ylim[0])

            ax.set_xlim([xdata - new_width * (1-relx), xdata + new_width * (relx)])
            ax.set_ylim([ydata - new_height * (1-rely), ydata + new_height * (rely)])
            ax.figure.canvas.draw()

        fig = ax.get_figure() # get the figure of interest
        fig.canvas.mpl_connect('scroll_event', zoom)

        return zoom

    def pan_factory(self, ax):
        def onPress(event):
            if event.inaxes != ax: return
            self.cur_xlim = ax.get_xlim()
            self.cur_ylim = ax.get_ylim()
            self.press = self.x0, self.y0, event.xdata, event.ydata
            self.x0, self.y0, self.xpress, self.ypress = self.press

        def onRelease(event):
            self.press = None
            ax.figure.canvas.draw()

        def onMotion(event):
            if self.press is None: return
            if event.inaxes != ax: return
            dx = event.xdata - self.xpress
            dy = event.ydata - self.ypress
            self.cur_xlim -= dx
            self.cur_ylim -= dy
            ax.set_xlim(self.cur_xlim)
```

```
194                 ax.set_ylim(self.cur_ylim)
195 
196                 ax.figure.canvas.draw()
197 
198             fig = ax.get_figure() # get the figure of interest
199 
200             # attach the call back
201             fig.canvas.mpl_connect('button_press_event', onPress)
202             fig.canvas.mpl_connect('button_release_event', onRelease)
203             fig.canvas.mpl_connect('motion_notify_event', onMotion)
204 
205             #return the function
206             return onMotion
207 
208 
209     if __name__ == '__main__':
210         corona_map = CoronaMap()
211         corona_map.displayCoMap("South Korea", "City", 'Reds', True, True, True)
```